Ensemble Learning

Nipun Batra and teaching staff

IIT Gandhinagar

August 14, 2025

Table of Contents

In this section

1. Introduction to Ensemble Learning

Ensemble Methods

Use multiple models for prediction.
Most winning entries of Kaggle competitions use ensemble learning.

Ensemble Methods

Use multiple models for prediction.
Most winning entries of Kaggle competitions use ensemble learning.

Example:
Classifier 1 - Good
Classifier 2 - Good
Classifier 3 - Bad

Using **majority voting**, we predict Good.

Ensemble Methods

Use multiple models for prediction.
Most winning entries of Kaggle competitions use ensemble learning.

Example:
Regressor 1 - 20
Regressor 2 - 30
Regressor 3 - 30

Using **averaging**, we predict $\frac{80}{3}$.

Intuition

Intuition

Based on *Ensemble Methods in Machine Learning* (Dietterich)

Three reasons why ensembles make sense:

Intuition

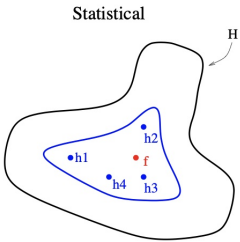
Based on *Ensemble Methods in Machine Learning* (Dietterich)

Three reasons why ensembles make sense:
1) Statistical: *Limited data* $\Rightarrow$ many competing hypotheses fit training equally well.

Intuition

Based on *Ensemble Methods in Machine Learning* (Dietterich)

Three reasons why ensembles make sense:
1) Statistical: *Limited data* $\Rightarrow$ many competing hypotheses fit training equally well.
E.g., many decision trees can reach similar train accuracy.



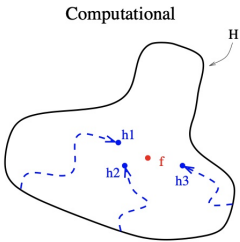
Intuition

Intuition

2) Computational: Some learners are *greedy/non-convex* and can get stuck in local optima.

Intuition

2) Computational: Some learners are *greedy/non-convex* and can get stuck in local optima.
E.g., decision trees employ greedy splits.



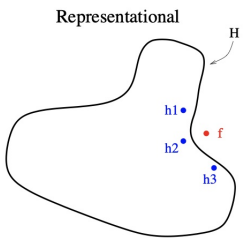
Intuition

Intuition

3) Representational: Some models cannot express the true decision boundary.

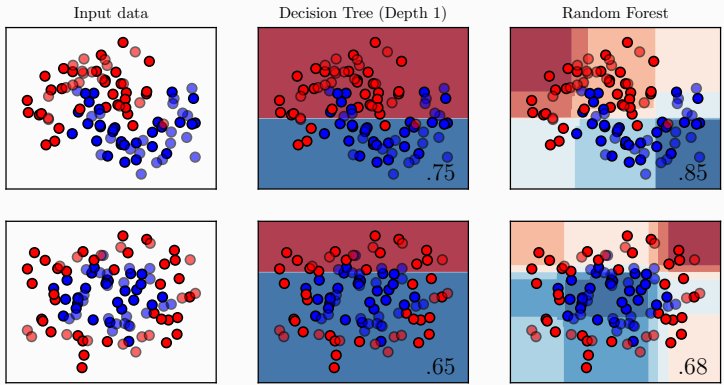
Intuition

3) Representational: Some models cannot express the true decision boundary.
E.g., axis-parallel splits only (vanilla decision trees).



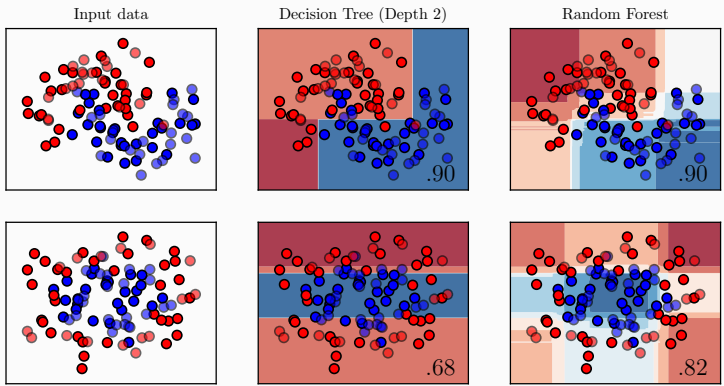
Representation of Limited Depth DTs vs RFs

Notebook: ensemble-representation.html



Representation of Limited Depth DTs vs RFs

Notebook: ensemble-representation.html



Necessary and Sufficient Conditions

Necessary and Sufficient Conditions

1) A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are **accurate and diverse**.

Necessary and Sufficient Conditions

1) A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are **accurate and diverse**.

2) An **accurate** classifier

Necessary and Sufficient Conditions

1) A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are **accurate and diverse**.

2) An **accurate** classifier is one that has an error rate better than random guessing on new x values.

Necessary and Sufficient Conditions

1) A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are **accurate and diverse**.

2) An **accurate** classifier is one that has an error rate better than random guessing on new x values.

3) Two classifiers are **diverse**

Necessary and Sufficient Conditions

1) A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are **accurate and diverse**.

2) An **accurate** classifier is one that has an error rate better than random guessing on new x values.

3) Two classifiers are **diverse** if they make different errors on new data points.

Necessary and Sufficient Conditions

Imagine we have an ensemble of three classifiers (h_1, h_2, h_3) and consider a new case x.

Necessary and Sufficient Conditions

Imagine we have an ensemble of three classifiers (h_1, h_2, h_3) and consider a new case x.
If the three classifiers are identical (not diverse), when $h_1(x)$ is wrong $h_2(x)$ and $h_3(x)$ will also be wrong.

Necessary and Sufficient Conditions

Imagine we have an ensemble of three classifiers (h_1, h_2, h_3) and consider a new case x.
If the three classifiers are identical (not diverse), when $h_1(x)$ is wrong $h_2(x)$ and $h_3(x)$ will also be wrong.
If errors are uncorrelated, when $h_1(x)$ is wrong, $h_2(x)$ and $h_3(x)$ may be correct, so a majority vote correctly classifies x.

Intuition for Ensembles: Quantitative View

Intuition for Ensembles: Quantitative View

Error probability of each model $\varepsilon = 0.3$

$Pr(\text{ensemble wrong}) = {}^{3}C_{2}\varepsilon^{2}(1-\varepsilon)^{1} + {}^{3}C_{3}\varepsilon^{3}(1-\varepsilon)^{0} = 0.19 \leq 0.3$

Some calculations

Number of Models	Ensemble Error	Individual Error
1	0.3	0.3
3	0.216	0.3
5	0.163	0.3

Some calculations

Number of Models	Ensemble Error	Individual Error
1	0.6	0.6
3	0.648	0.6
5	0.683	0.6

Ensemble Methods

Where does ensemble learning not work well?

Ensemble Methods

Where does ensemble learning not work well?

- The base model is bad.
- All models make similar/ correlated predictions.

Bagging

Also known as **Bootstrap Aggregation**.

Bagging

Also known as **Bootstrap Aggregation**.
Key idea: Reduce **variance**.

Bagging

Also known as **Bootstrap Aggregation**.
Key idea: Reduce **variance**.

How to learn different classifiers from the same data?

Bagging

Also known as **Bootstrap Aggregation**.
Key idea: Reduce **variance**.

How to learn different classifiers from the same data?

Think about **resampling** (cf. cross-validation)!

Bagging

Also known as **Bootstrap Aggregation**.
Key idea: Reduce **variance**.

How to learn different classifiers from the same data?

Think about **resampling** (cf. cross-validation)!

Create multiple datasets using **sampling with replacement**.

Bagging

Dataset with n samples: $D_1, \ldots, D_n$.
For each model, create a new dataset of size n by sampling uniformly with replacement.

Bagging

Dataset with n samples: $D_1, \ldots, D_n$.
For each model, create a new dataset of size n by sampling uniformly with replacement.

Round 1: $D_1, D_3, D_6, D_1, \ldots, D_n$
Round 2: $D_2, D_4, D_1, D_{80}, \ldots, D_3$
$\vdots$

Bagging

Dataset with n samples: $D_1, \ldots, D_n$.
For each model, create a new dataset of size n by sampling uniformly with replacement.

Round 1: $D_1, D_3, D_6, D_1, \ldots, D_n$
Round 2: $D_2, D_4, D_1, D_{80}, \ldots, D_3$
$\vdots$
Repetition of samples is possible.

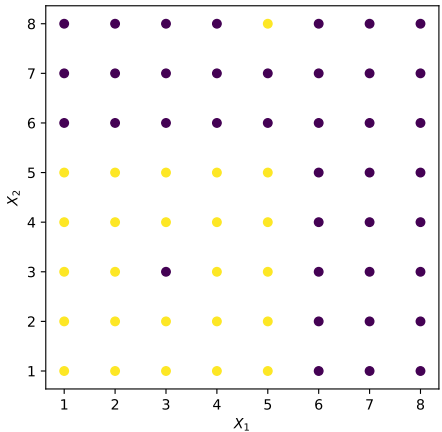
Bagging

Dataset with n samples: $D_1, \ldots, D_n$.
For each model, create a new dataset of size n by sampling uniformly with replacement.

Round 1: $D_1, D_3, D_6, D_1, \ldots, D_n$
Round 2: $D_2, D_4, D_1, D_{80}, \ldots, D_3$
$\vdots$
Repetition of samples is possible.
Train the same model on each "bag" and **aggregate** predictions.

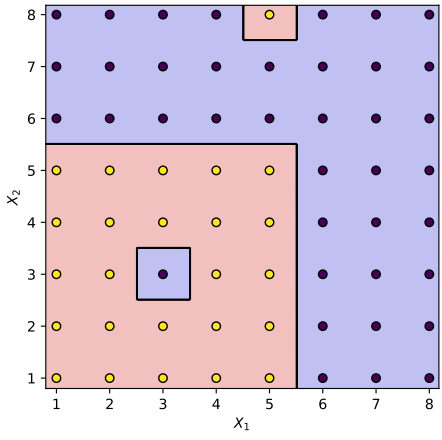
Bagging : Classification Example

Consider the dataset below. Points (3,3) and (5,8) are anomalies.



Bagging : Classification Example

Decision Boundary for decision tree with depth 6.



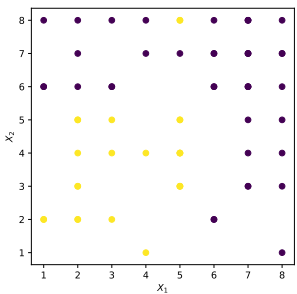
Bagging : Classification Example

Let's use bagging with an ensemble of 5 trees.

Bagging : Classification Example

Let's use bagging with an ensemble of 5 trees.

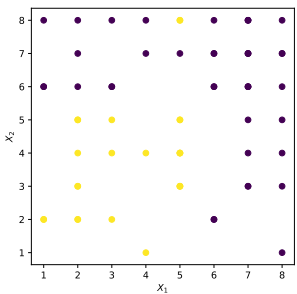
Round - 1



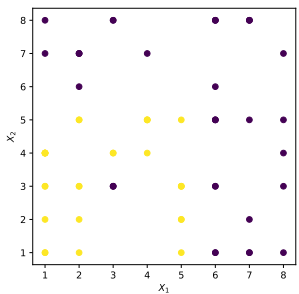
Bagging : Classification Example

Let's use bagging with an ensemble of 5 trees.

Round - 1



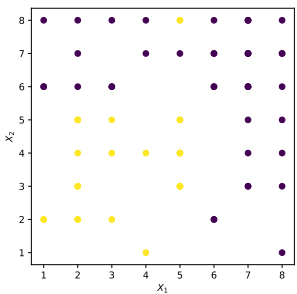
Round - 2



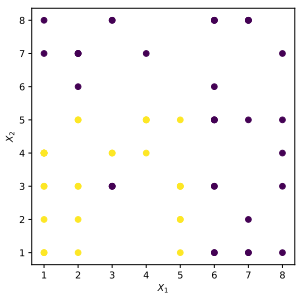
Bagging : Classification Example

Let's use bagging with an ensemble of 5 trees.

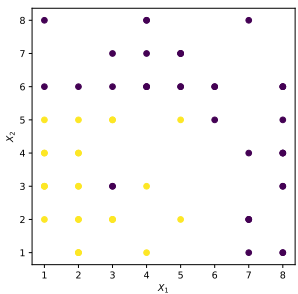
Round - 1



Round - 2



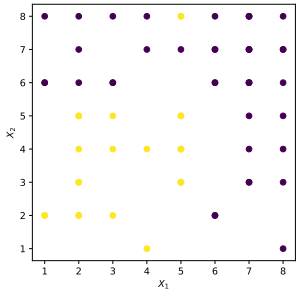
Round - 3



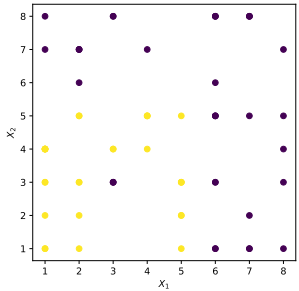
Bagging : Classification Example

Let's use bagging with an ensemble of 5 trees.

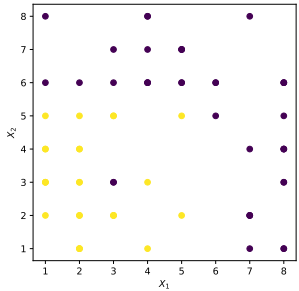
Round - 1



Round - 2



Round - 3

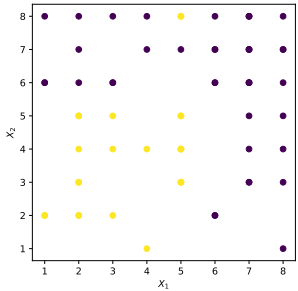


Round - 4

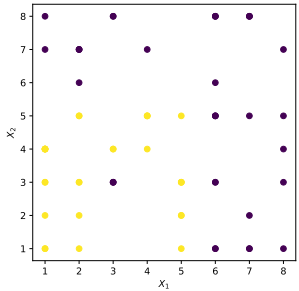
Bagging : Classification Example

Let's use bagging with an ensemble of 5 trees.

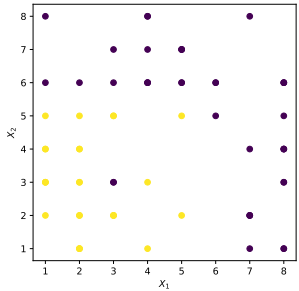
Round - 1



Round - 2



Round - 3



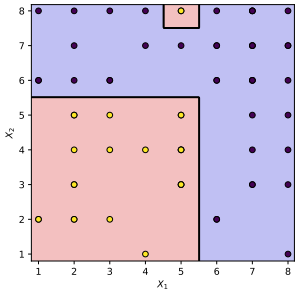
Round - 4

Round - 5

Bagging : Classification Example

Bagging : Classification Example

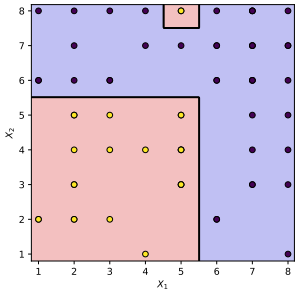
Round - 1



Tree Depth = 4

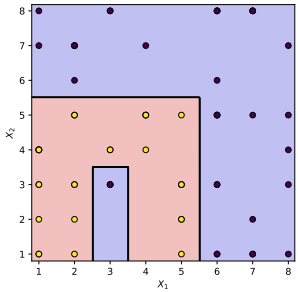
Bagging : Classification Example

Round - 1



Tree Depth = 4

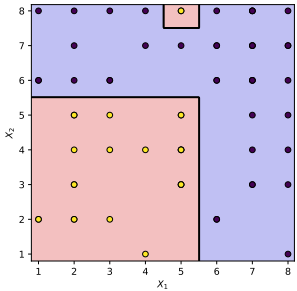
Round - 2



Tree Depth = 5

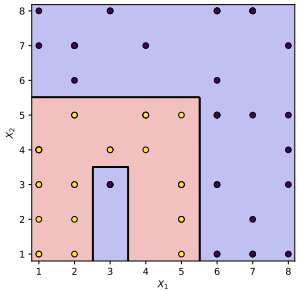
Bagging : Classification Example

Round - 1



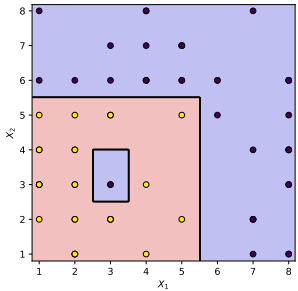
Tree Depth = 4

Round - 2



Tree Depth = 5

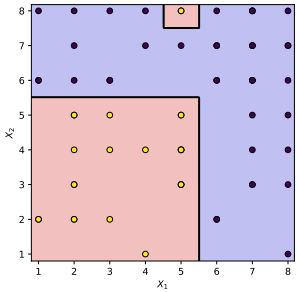
Round - 3



Tree Depth = 5

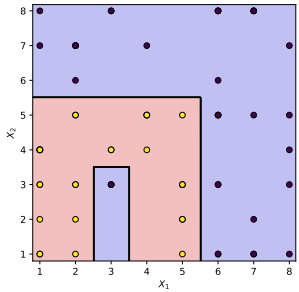
Bagging : Classification Example

Round - 1



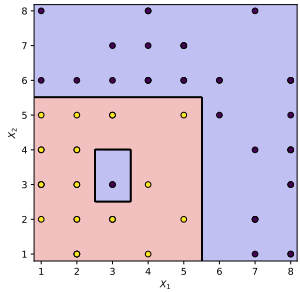
Tree Depth = 4

Round - 2



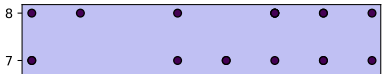
Tree Depth = 5

Round - 3



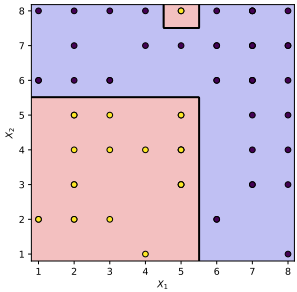
Tree Depth = 5

Round - 4



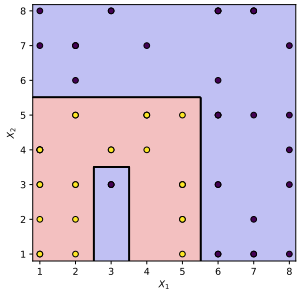
Bagging : Classification Example

Round - 1



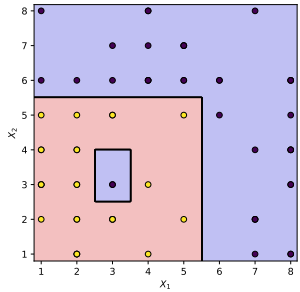
Tree Depth = 4

Round - 2



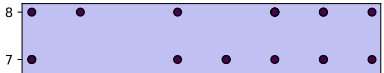
Tree Depth = 5

Round - 3

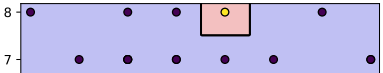


Tree Depth = 5

Round - 4

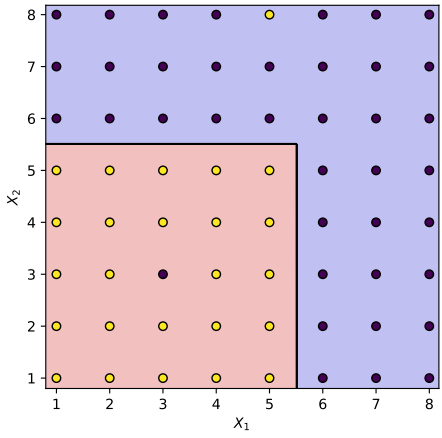


Round - 5



Bagging : Classification Example

Using majority voting to combine all predictions, we get:



Bagging

Summary

- Combine **strong** learners to reduce **variance**.
- Learners are trained **independently** on bootstrap samples.

Boosting

- Combine **weak** learners to reduce **bias**.

Boosting

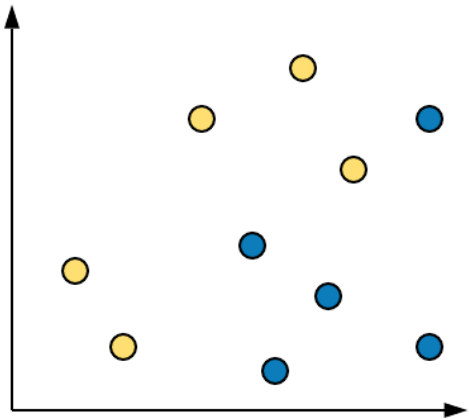
- Combine **weak** learners to reduce **bias**.
- Learners are built **incrementally**.

Boosting

- Combine **weak** learners to reduce **bias**.
- Learners are built **incrementally**.
- Each round focuses on **harder** samples (reweighted).

Boosting : AdaBoost

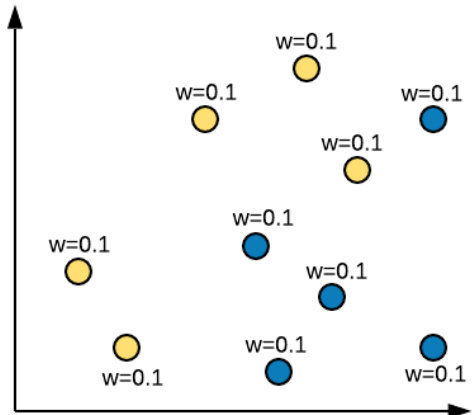
Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.



Boosting : AdaBoost

Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

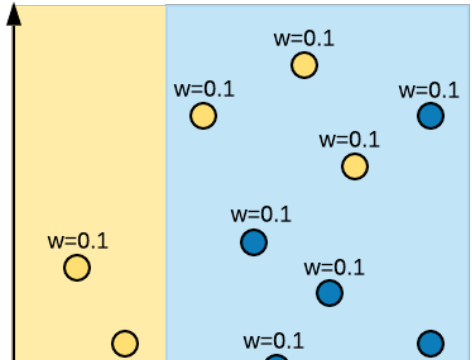
1. Initialize weights: $w_i = \frac{1}{N}$



Boosting : AdaBoost

Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

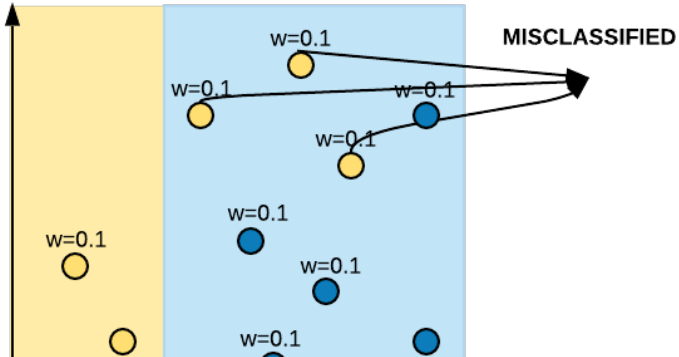
1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i



Boosting : AdaBoost

Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

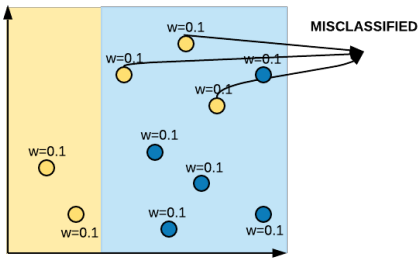
1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i



Boosting : AdaBoost

Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i
   2) Compute weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$

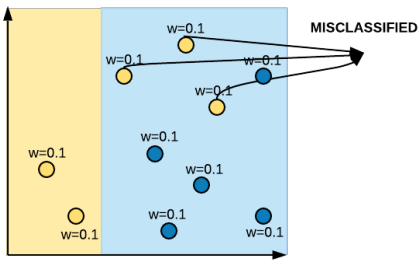


$$err_1 = \frac{0.3}{1}$$

Boosting : AdaBoost

Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i
   2) Compute weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2}\log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$



$err_1 = 0.3$

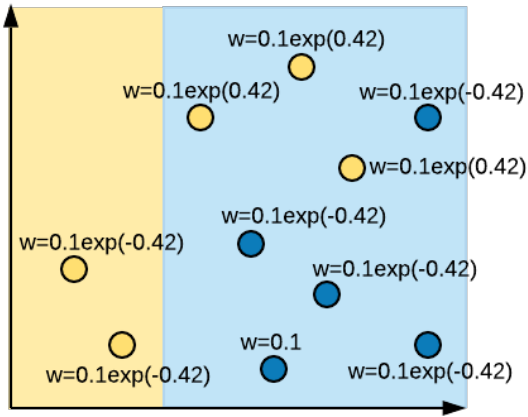
$\alpha_1 = \frac{1}{2}\log\left(\frac{0.7}{0.3}\right) \approx 0.42$

Boosting : AdaBoost

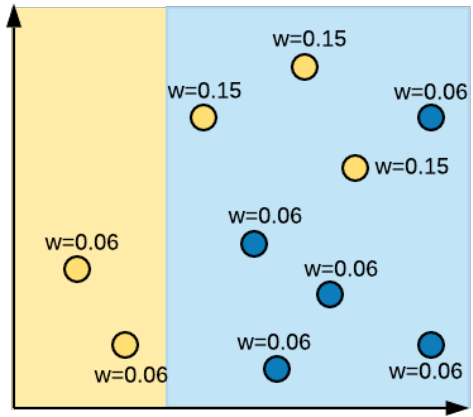
Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i
   2) Compute weighted error: $\text{err}_m = \dfrac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2} \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
   4) If correct: $w_i \leftarrow w_i e^{-\alpha_m}$ If incorrect: $w_i \leftarrow w_i e^{\alpha_m}$

Boosting : AdaBoost



Boosting : AdaBoost

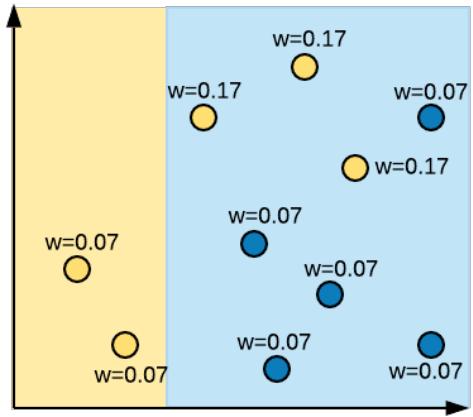


Boosting : AdaBoost

Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i
   2) Compute weighted error: $\text{err}_m = \dfrac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2} \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
   4) Update w_i as above
   5) Normalize w_i to sum to 1

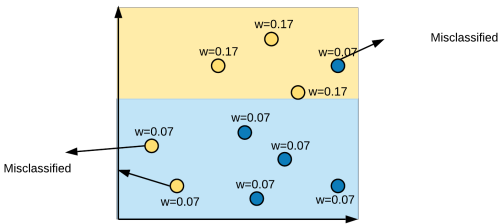
Boosting : AdaBoost



Boosting : AdaBoost

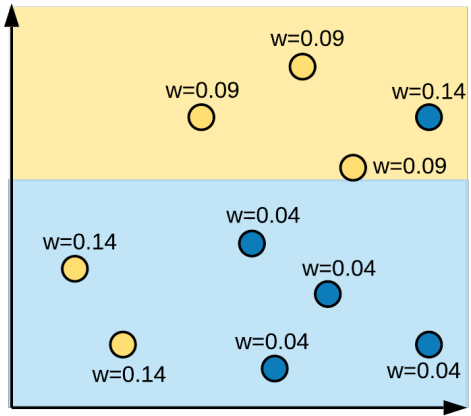
Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i
   2) Compute weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2} \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$



$$err_2 = 0.21$$
$$\alpha_2 = \frac{1}{2} \log\left(\frac{0.79}{0.21}\right) \approx 0.66$$

Boosting : AdaBoost

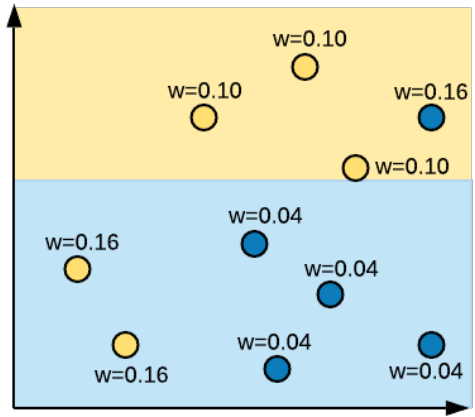


Boosting : AdaBoost

Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i
   2) Compute weighted error: $\text{err}_m = \frac{\sum_i w_i (\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2} \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
   4) Update w_i as above

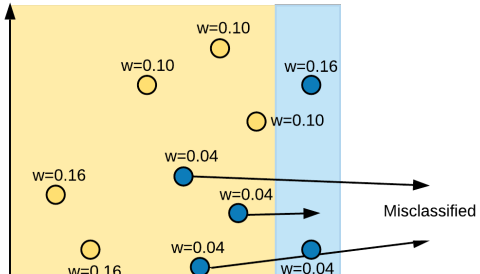
Boosting : AdaBoost



Boosting : AdaBoost

Consider a dataset of N samples. Sample i has weight w_i. There are M classifiers.

1. Initialize weights: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i
   2) Compute weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2} \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$



$err_3 = 0.12$

$\alpha_3 = \frac{1}{2} \log\left(\frac{0.88}{0.12}\right) \approx 0.99$

Boosting: AdaBoost

Intuition: after each iteration, importance of wrongly classified samples increases (weights up), and importance of correctly classified samples decreases (weights down).

Boosting: AdaBoost

Testing

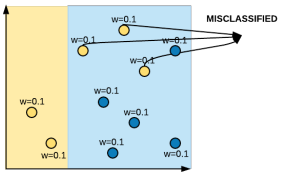
- For each sample x, compute $h_m(x)$ for all m.
- Final prediction: $\mathrm{sign}(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_M h_M(x))$.

Boosting: AdaBoost

Example

Boosting: AdaBoost

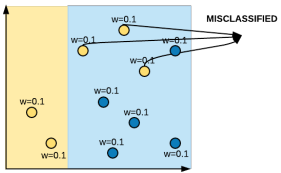
Example



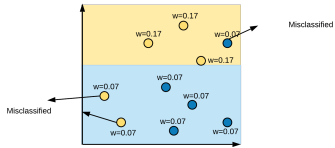
$\alpha_1 = 0.42$

Boosting: AdaBoost

Example



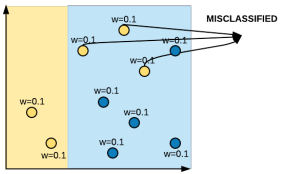
$\alpha_1 = 0.42$



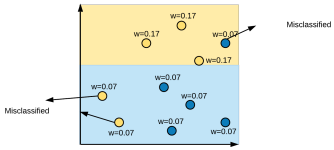
$\alpha_2 = 0.66$

Boosting: AdaBoost

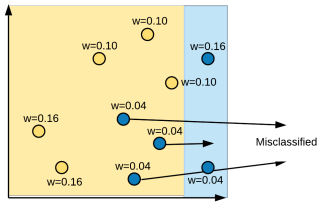
Example



$\alpha_1 = 0.42$



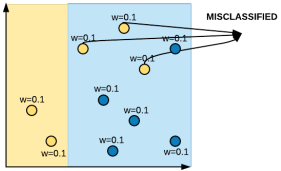
$\alpha_2 = 0.66$



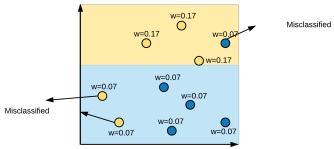
$\alpha_3 = 0.99$

Boosting: AdaBoost

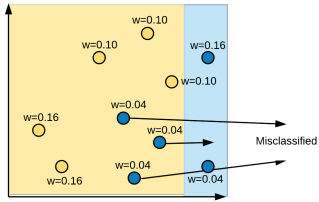
Example



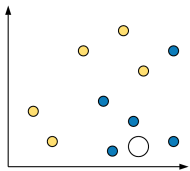
$\alpha_1 = 0.42$



$\alpha_2 = 0.66$

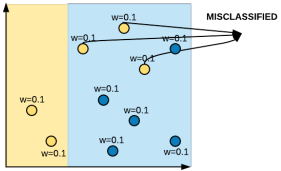


$\alpha_3 = 0.99$

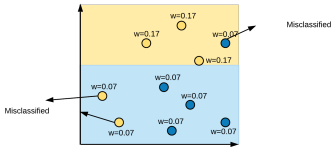


Boosting: AdaBoost

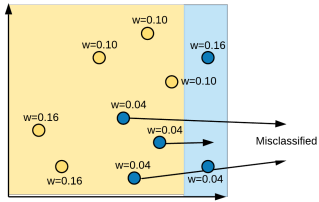
Example



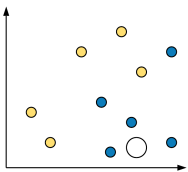
$\alpha_1 = 0.42$



$\alpha_2 = 0.66$



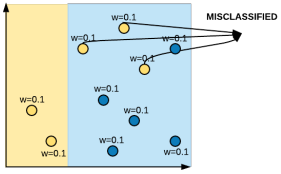
$\alpha_3 = 0.99$



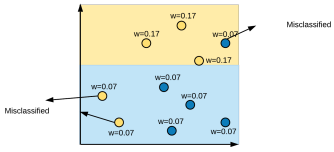
Let yellow be $+1$ and blue be -1.

Boosting: AdaBoost

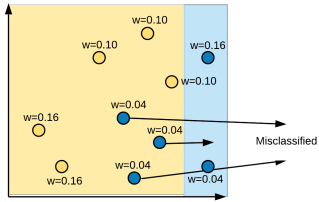
Example



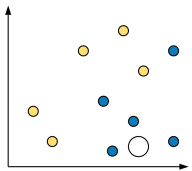
$\alpha_1 = 0.42$



$\alpha_2 = 0.66$



$\alpha_3 = 0.99$



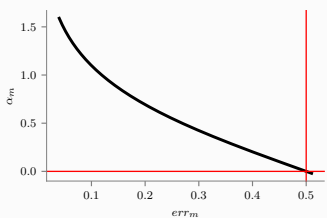
Let yellow be $+1$ and blue be -1.

Prediction
$= \text{sign}(0.42 \cdot -1 + 0.66 \cdot -1 + 0.99 \cdot +1) =$
Negative (blue).

Intuition behind weight update formula

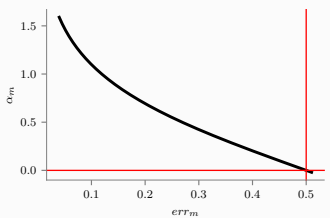
Intuition behind weight update formula

Notebook: boosting-explanation.html

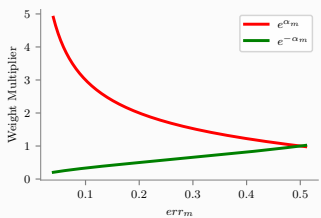


Intuition behind weight update formula

Notebook: boosting-explanation.html



Notebook: boosting-explanation.html



AdaBoost for Regression

From paper: *Improving Regressors using Boosting Techniques*

Our problem will be that the modeling error is also nonzero because we have to determine the model in the presence of noise. Since we don't know the probability distributions, we approximate the expectation of the ME and PE using the sample ME (if the truth is known) and sample PE and then average over multiple experiments.

In the following discussion, we detail both bagging and boosting. We then discuss how to build trees which are the basic building blocks of our regression machines and use these ensembles on some standard test functions.

2. BAGGING

The following is a paraphrase of Breiman (1996b) with some difference in notation. Suppose we pick with replacement N_1 examples from the training set of size N_1 and call the k'th set of observations O_k. Based on these observations, we form a predictor $y^{[O_k]}(\mathbf{x},O_k)$. Because we are sampling with replacement, we may have multiple observations or no observations of a particular training example. Sampling with replacement is sometimes termed bootstrap sampling [Efron and Tibshirani (1993)] and therefore this method is called **b**ootstrap **agg**regating or bagging for short. The ensemble predictor is formed from the approximation to the expectation over all the observation sets, i.e. $E_O[y^{[O]}(\mathbf{x},O)]$ by using the average of the outputs of all the predictors. Breiman discusses which algorithms are good candidates for predictors and concludes that the best predictors are unstable, i.e., a small change in the training set O_k causes a large change in the predictor $y^{[O]}(\mathbf{x},O_k)$. Good candidates are regression trees and neural nets.

3. BOOSTING

In bagging, each training example is equally likely to be picked. In boosting, the probability of a particular example being in the training set of a particular machine depends on the performance of the prior machines on that example. The following is a modification of *AdaBoost.R* [Freund and Schapire (1996a)].

Initially, to each training pattern we assign a weight $w_i=1 \quad i=1,...,N_1$

Repeat the following while the average loss $\bar{L}$ defined

set. Each machine makes a hypothesis: $h_t: x \to y$

3. Pass *every* member of the training set through this machine to obtain a prediction $y_i^{[t]}(\mathbf{x}_i) \quad i=1,..N_1$.

4. Calculate a loss for each training sample $L_i = L\left[\,|\,y_i^{[t]}(\mathbf{x}_i)-y_i\,|\,\right]$. The loss L may be of any functional form as long as $L\in[0,1]$. If we let

$$D=sup\,|\,y_i^{[t]}(\mathbf{x}_i)-y_i\,| \quad i=1,...,N_1$$

then we have three candidate loss functions:

$$L_i=\frac{|\,y_i^{[t]}(\mathbf{x}_i)-y_i\,|}{D} \qquad (linear)$$

$$L_i=\frac{|\,y_i^{[t]}(\mathbf{x}_i)-y_i\,|^2}{D^2} \qquad (square\ law)$$

$$L_i=1-exp\left[\frac{-|\,y_i^{[t]}(\mathbf{x}_i)-y_i\,|}{D}\right] \qquad (exponential)$$

5. Calculate an average loss: $\bar{L}=\sum_{i=1}^{N_1} L_i p_i$

6. Form $\beta=\frac{\bar{L}}{1-\bar{L}}$. β is a measure of confidence in the predictor. Low β means high confidence in the prediction.

7. Update the weights: $w_i \to w_i \beta^{**}[1-L_i]$, where ** indicates exponentiation. The smaller the loss, the more the weight is reduced making the probability smaller that this pattern will be picked as a member of the training set for the next machine in the ensemble.

8. For a particular input $\mathbf{x}_i$, each of the T machines makes a prediction h_t, $t=1,...,T$. Obtain the cumulative prediction h_f using the T predictors:

Random Forest

- Random Forest is an ensemble of decision trees.
- Two types of bagging: bootstrap (on data) and random subspace (on features).
- Random feature subsampling decorrelates trees $\Rightarrow$ reduces variance.

Random Forest

There are 3 parameters while training a random forest: **number of trees**, **number of features (m)**, **maximum depth**.

Training Algorithm

- For i^{th} tree ($i \in \{1 \cdots N\}$), select n samples from total N samples **with replacement**.
- Learn a decision tree on selected samples for the i^{th} round.

Learning Decision Tree (for RF)

- For each split, select m features from total M features and choose the best split only among those m.

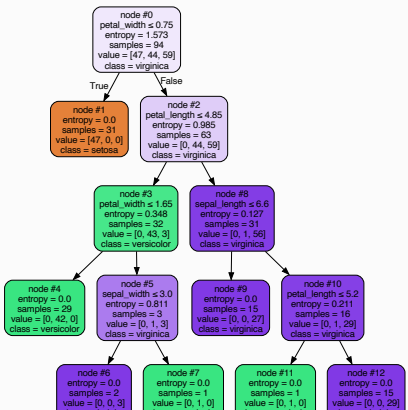
Dataset

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

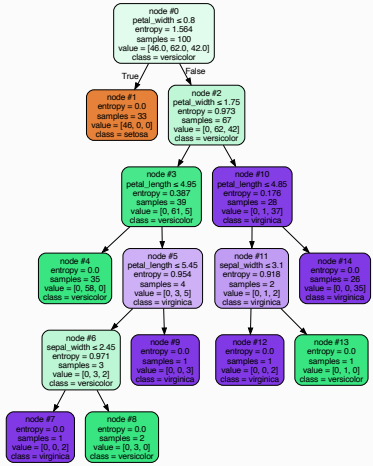
Decision Tree # 0

Notebook: ensemble-feature-importance.html



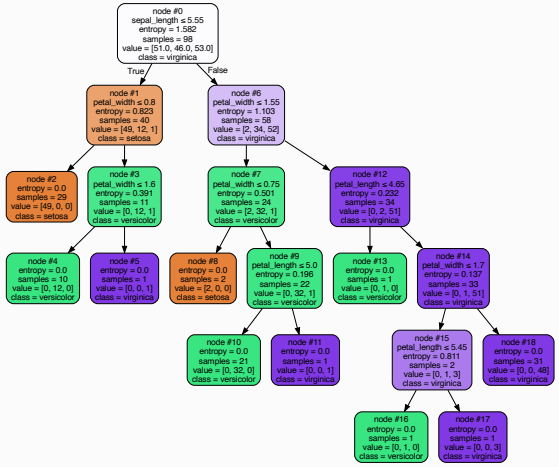
Decision Tree # 1

Notebook: ensemble-feature-importance.html



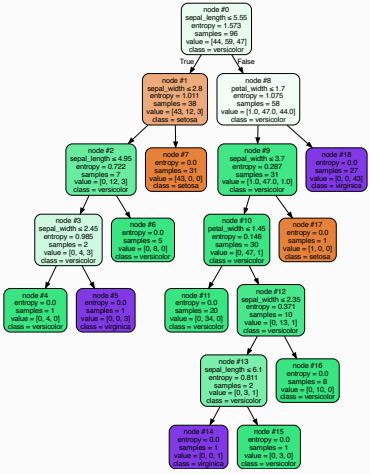
Decision Tree # 2

Notebook: ensemble-feature-importance.html



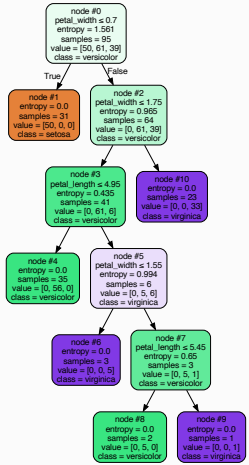
Decision Tree # 3

Notebook: ensemble-feature-importance.html



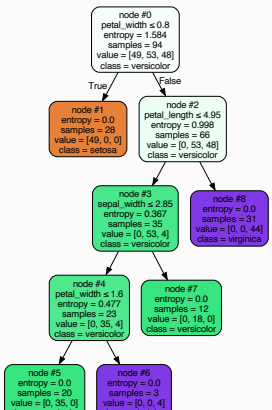
Decision Tree # 4

Notebook: ensemble-feature-importance.html



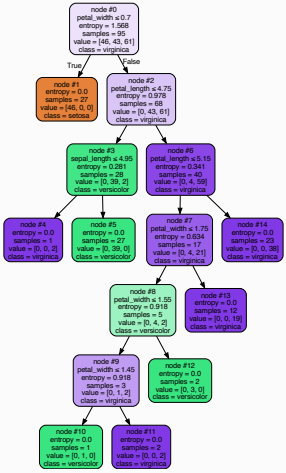
Decision Tree # 5

Notebook: ensemble-feature-importance.html



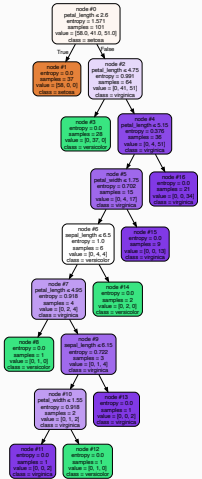
Decision Tree # 6

Notebook: ensemble-feature-importance.html



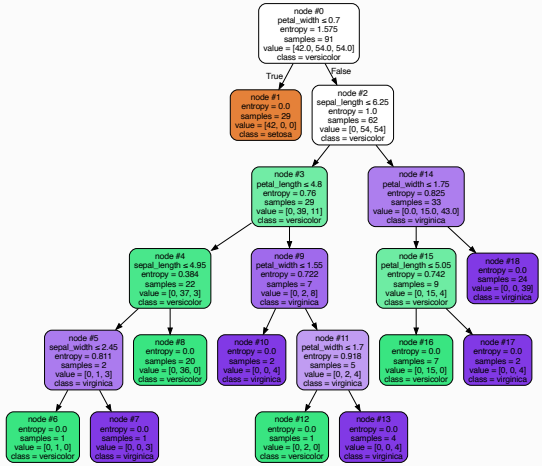
Decision Tree # 7

Notebook: ensemble-feature-importance.html



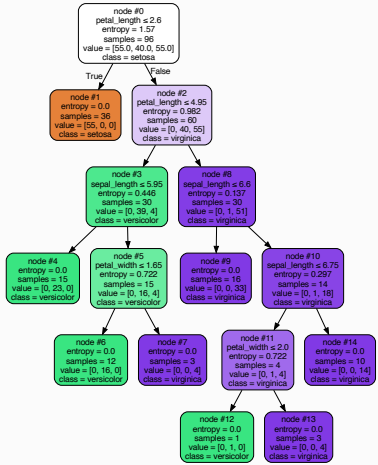
Decision Tree # 8

Notebook: ensemble-feature-importance.html

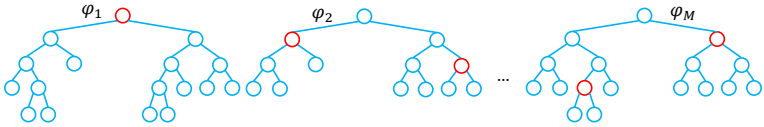


Decision Tree # 9

Notebook: ensemble-feature-importance.html



Feature Importance[1]



Importance of variable X_j for an ensemble of M trees φ_m is:

$$\text{Imp}(X_j) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t \in \varphi_m} 1(j_t = j) \Big[p(t)\, \Delta i(t) \Big],$$

where j_t denotes the variable used at node t, $p(t) = N_t / N$, and $\Delta i(t) = i(t) - \frac{N_{t_L}}{N_t} i(t_L) - \frac{N_{t_R}}{N_t} i(t_R)$.

[1] Slide Courtesy Gilles Louppe

Computed Feature Importance

Notebook: ensemble-feature-importance.html